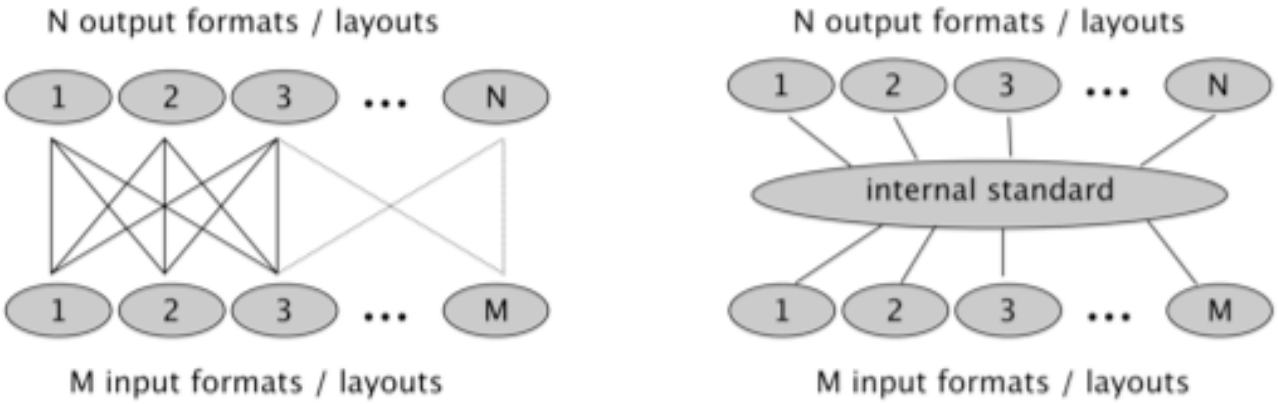
Mission Independent Representations: Time & Spacecraft Position

VHO Meeting, Aug. 31, 2007

Jon Vandegriff
JHU/APL

What is a mission independent representation?



Data unification involves the transformation of M different possible input formats (and layouts within those formats!) into the N different preferred formats of every scientist. In the past, each scientist creates the M different readers to ingest the data into a preferred output format. As a community-wide solution, this approach requires an inordinate amount of work, because M times N pieces of software must be written (see Fig. 2a).

A much more efficient approach, illustrated in Fig. 2b, is to create an intermediate, internal data arrangement to which all of the M incoming datasets are converted. This then requires M different readers. Once a dataset has been read in and converted to the single, intermediate internal arrangement, it has been unified (it exists within identical structures in the software – it has not yet been written out to a new file). Then each output mechanism can start with the unified version of the data, and so only N different output modules can provide mappings to each of the desired output formats. The overall complexity of this solution is not M times N, but just M plus N.

A system with this intermediate layer decouples the input datasets from the tools that will use the data (such as output modules). This decoupling creates a system that is much more easily expandable, since adding a new dataset requires a fixed amount of work – simply write one new reader, and all the output mechanisms will automatically be able to use it. Also, any new output mechanism or analysis tool will be based on the standardized internal representations, and therefore will be able to access any of the datasets for which a unifying reader exists.

Representing Time

1. overall goal:
   every reader must provide time in a recognized format and in a recognized time system (UTC, ET, TAI)
2. format: time as a single double precision value
3. time system: must use a uniform time system with no leap seconds!
   Options:
   - TAI (basically an offset UTC minus the leapseconds)
   - ET (seconds past J2000 start of noon on Jan 1, 2000)
4. this will be a **required** output of all readers
5. its not an overly onerous requirement to ask a reader to convert native times to our time format/system - this can be built into our generic readers

Representing S/C Position

1. there are a lot of different coordiante systems (reference frames) out there
2. in contrast to the time conversion, coordinate conversion is too difficult to be built into every reader
3. so the unified interface must accept a set of ref. frames
4. it should be a somewhat smallish set
5. we need to be careful about ambiguous coordinate system definitions
6. sample set of coordinate system interfaces is shown next
7. key idea: treat ephem info slightly differently: ask for ephem. data as a function rather than a table

```java
public interface IEphemerisDataLoader
{
    /** @return string describing the spacecraft label*/
    public String getSCID();

    /** @return String describing the sun ID */
    public String getSunID();

    /** @return Strings describing the bodies involved in this ephemeris */
    public String[] getBodyIDs();

    /** @return EphemInfo for the given <code>ephemIndex</code>*/
    public EphemInfo getEphems(int ephemIndex);

    /** @return number of available ephemerides for this loader*/
    public int getNumAvailableEphems();

    /** @return IEphemFunc for the given <code>info</code>*/
    public IEphemFunc getEphemerisFunction(EphemInfo info);

    /** @param center
     * @param target
     * @param frame
     * @return a function yielding ephemeris data at every point */
    public IEphemFunc getEphemerisFunction(String center, String target,
                                           String frame);
}
```

```
/**
 * EphemInfo - describes a particular ephemeris configuration
*
 */
public class EphemInfo
{
    private String center;

    private String target;

    private double[][] timeRanges;

    private boolean hasVelocity;

    private String frame;

    private String infoSource;
```

(getters and setters for all of these fields...)

```
}
```

```java
/**
 * Function interface for retrieving Ephemeris data
 */
public interface IEphemFunc
{
    /**
     * Gets the position for the given time <code>t</code>.
     * @param t time to calculate the position
     * @return double array giving the 3 position components
     */
    public double[] getPosition(double t);

    /**
     * Gets the velocity for the given time <code>t</code>.  If no velocity
     * information is available (as specified by the <code>hasVelocity</code>
     * flag in the EphemInfo objects, this returns null;
     * @param t time to calculate the velocity
     * @return double array giving the 3 velocity components; null if no
     * velocity information is available in the ephemeris
     */
    public double[] getVelocity(double t);

    /**
     * @return EphemInfo object describing this ephemeris
     */
    public EphemInfo getEphemInfo();
}
```